

The YPA - An Assistant for Classified Directory Enquiries

Anne De Roeck¹, Udo Kruschwitz¹, Paul Scott¹, Sam Steel¹, Ray Turner¹, and Nick Webb²

¹ Department of Computer Science, University of Essex,
Wivenhoe Park, Colchester,
CO4 3SQ, United Kingdom
{deroe, udo, scotp, sam, turnr}@essex.ac.uk

² Department of Computer Science, University of Sheffield,
Regent Court, 211 Portbello Street,
Sheffield S1 4DP, United Kingdom
N.Webb@sheffield.ac.uk

Abstract. The YPA is a directory enquiry system which allows a user to access advertiser information in classified directories [1]. It converts *semi-structured data* in the *Yellow Pages*¹ machine readable classified directories into a set of indices appropriate to the domain and task, and converts natural language queries into filled slot and filler structures appropriate for queries in the domain. The generation of answers requires a domain-dependent *query construction* step, connecting the indices and the slot and fillers. The YPA illustrates an unusual but useful intermediate point between information retrieval and logical knowledge representation.

1 Introduction

The YPA stands mid-way between full natural language analysis and information retrieval. It does not attempt to build a deep representation of the facts in the Yellow Pages. However, it goes considerably beyond keyword matching. It does not treat all parts of a query alike, and it recognizes different functions of different parts of the input text. It uses an on-line thesaurus *WordNet* [2] and cross-referencing to connect tokens in a query with tokens in the Yellow Pages data. Different forms of a word are reduced to a base form where this is straightforward.

The functionality it offers is intended to assist a Yellow Pages customer, or, perhaps more realistically, a Talking Pages operator. Talking Pages is a service in which a customer rings a call-centre operator and asks a question which the operator then attempts to answer from a Yellow Pages like classified directory.

¹ Yellow Pages® and Talking Pages® are registered trade marks of British Telecommunications plc in the United Kingdom

The goal is to narrow, rather than completely close, the gap between a user's enquiry and the Yellow Pages data.

The system is a mixed initiative system. It allows free text input, and uses robust, shallow, parsing to extract the important parts of a query. Free text query refinements are also possible later. However, when the system has a choice of possible replies, or sees a range of obvious possible query refinements, it may offer the user a range of options it has collected.

Though the system is described very much in the light of its applications to Yellow Pages, we believe that the techniques, both of constructing the *Backend* indices and of handling the *Frontend* interaction, will be applicable to many similar domains involving interactions with semi-structured data.

The paper first discusses what is needed from a Yellow Pages assistant (section 2) followed by a system overview (section 3), which also describes the function of each of the modules by reference to a running example. Section 4 deals with our most recent experiences including system evaluation. Finally section 5 deals with conclusions and future work.

2 Motivation and Related Work

Finding information in a classified directory can be straightforward but requires prior knowledge of the indexing system employed by that directory. Take the example:

I want an Italian restaurant in Colchester that takes Visa.

Clearly the easiest route would be to take a local classified directory and look under the heading *Italian Restaurants* for the required addresses. However, several problems can occur, among them:

- There is no heading *Italian Restaurants* and under *Restaurants* the list of advertisements is very long but contains hardly any offering *Italian cuisine*.
- There are no *Italian restaurants* at all.
- None of the advertisements specifically say that they accept *Visa*.
- The best matches might be found under a heading *Pizzerias*.
- There could be an *Italian Restaurant* in *Wivenhoe*, just 3 miles outside *Colchester* which accepts *Visa* and says so.
- *Colchester* is not covered by the local directory.

The task of the YPA is to cut across the headings, allowing the user direct access to those advertisements which match their individual requirements. This task is made difficult by the problem of identifying within a query what it is that is of interest to a user is. For example:

I want a good Italian. - We are interested in *Italian*.

I want a pizza service. - We are interested in *pizza*.

In order to do this, we have to “narrow the gap” between the requirements of the user and the indices offered by the *Yellow Pages* domain.

From the user’s end, we need to offer some natural language front end to soften the interface, coupled with some easily obtained common sense knowledge to reason about possible extensions to the user’s input. From the *Yellow Pages* domain, we build “knowledge-based” indices, capturing more accurately the nature of the information stored within individual advertisements.

Together these techniques will give us more power than simple information retrieval techniques, which although powerful in the right context would still be unable to find the entry *The Pasta Palace* under heading *Take Away Food* from an enquiry about *Italian restaurants*. In addition, our knowledge base will be easier to build and maintain than one built up through deep knowledge representation, using logic and theorem proving.

If we consider the YPA as a natural language dialogue system, we find it comparable to numerous previous systems. In the main these sophisticated systems (for example OVIS [3], The Philips train timetable system [4], Sundial [5, 6], TRAINS [7] and Verbmobil [8]) are concerned with time and schedule information. These are heavily restricted domains, in terms of size and complexity of data. In addition, there are a number of systems which retrieve addresses from *Yellow Pages*. Most notable of these are *Voyager* (and its sibling, *Galaxy*) [9, 10] and IDAS², an interactive directory assistance project funded by the European Union. IDAS has as a goal the effective disambiguation of user queries, and the narrowing of the search space of the query. This project is still in the development phase. *Voyager* and *Galaxy* have been around for some time, but the implementations are restricted to sample domains or small-scale address databases (e.g. 150 objects in the *Voyager* system [11]). What seems to be the greatest commonality between these systems and the YPA is the fact that an interaction involves the system filling a set of slots — for instance, in a travel domain, start and finish time and place — and then reacting appropriately to the set of filled slots.

A separate issue is the *Backend* construction process which takes the raw data (the so-called *YP printing tape*) and creates a database that retains as much information and relations as possible for the online enquiry system. This was described in detail in De Roeck *et al.* [1]. The input data is *semi-structured* in a sense that a record structure for the addresses and headings does exist but the internal structure of these entries is not formally defined. Usually this consists of partial English sentences, address information, telephone patterns, etc. Several types of advertisements exist: *free*, *line* and *semi-display* entries with *semi-display* advertisements carrying more information than just the name and the address of the advertiser. Standard *Information Retrieval* approaches do not help because the addresses are too short. Here is a typical *semi-display* entry as it appears in the printed directory (in this case listed under *Hotels & Inns*):

² <http://www.linglink.lu/le/projects/idas>

UDO, THE

TOWN CENTRE BED & BREAKFAST
EN SUITE ROOMS-AMPLE PARKING
High St,Ipswich,01473 123456

The actual source code in the *printing tape* does not give us much more structured information than what can be seen in the printed advertisement:

```
195SS15SBZS9810289N955030800 0 0150UDO,THE^
195SS15SBZS9810289B935020800C0      TOWN CENTRE BED & BREAKFAST^
195SS15SBZS9810289B935020800C0      EN SUITE ROOMS-AMPLE PARKING^
195SS15SBZS9810289B935020800C0      CHigh St,Ipswich,01473\$_123456^
```

The columns of coding on the left have considerable structure but are almost entirely about accounting and how the lines are to appear when printed. They contain very little about what is being advertised: that is almost entirely in the marked-up advertisement text on the right.

When comparing this data with a typical user request it is often hard to find advertisements which do actually satisfy the complete user query. The user might have asked for *hotels with en suite bathrooms*, something which cannot be found in the list of advertisements. But the above example is still a very good match.

We see the task of the dialogue system as narrowing the gap between a user request and what the index database of the system can supply.

Thus, the input data must be transformed into an appropriate representation which does allow matching of user queries to advertisements as effectively as possible combining various sorts of information like the name of the heading, the keywords in the free text of an entry (*TOWN CENTRE BED & BREAKFAST ...*), etc.

In order to obtain such a *Backend* database of indices we can distinguish two extreme positions for the construction process:

- *Knowledge Representation*, i.e. express the data in logic and do theorem proving in the online YPA system.
- *Information Retrieval*, i.e. use morphology and string matching.

There are at least two approaches for this sort of problem that should be mentioned here. *Publishers Depot* [12] is a commercial product that allows the retrieval of pictures based on the corresponding caption.³ That means the documents are very short compared to standard *IR* tasks, just like in our case. On the other hand there has been work on *conceptual indexing* [13]. A knowledge base is built that is based on the observation that there are few true synonyms in the

³ <http://www.picturequest.com>

language and usually there is a generality relationship like: *car* \rightarrow *automobile* \rightarrow *motor vehicle* which can be expressed as a *subsumption* relationship. Woods [13] employs an example from the *Yellow Pages*:

- The *Yellow Pages* do not contain a heading “automobile steam cleaning”.
- There are basic facts (axioms): (1) a *car* is an *automobile* and (2) *washing* is some kind of *cleaning*.
- The algorithm has to infer that *car washing* is some sort of *automobile cleaning*.

Such a knowledge base has been applied to precision content retrieval knowing that standard *Information Retrieval* approaches are not very effective for *short queries* [14].

Our approach can be placed somewhere between the one used by Flank [12] and Woods [13].

A number of online directory enquiry systems are actually in use. Some of the existing systems that offer information one would expect from the *Yellow Pages* are: *Electronic Yellow Pages (U.K.)*, *Scoot (U.K.)*⁴, *NYNEX Yellow Pages (U.S.A)*⁵, *BigBook (U.S.A)*⁶ and *Switchboard (U.S.A)*⁷. This type of commercial system functions quite differently from traditional *NLP* or *IR* systems. While they access large address databases of the same sort that we deal with, they all share a number of limitations:

- a very flat *Frontend* which offers at most pattern matching;
- a very simple lookup database which does not permit the access of free text hidden in the addresses;
- the inability to cope with words not found in the database (i.e. in the categories or names).

The intermediate position that we have chosen extracts relatively flat relations from the incoming data and uses text-retrieval-like methods for this *offline* process, while we can still apply theorem-proving-like methods in the *online* system.

3 Overall Structure of the YPA

The YPA is an interactive system. A conversation cycle with the YPA can be roughly described as follows. A user utterance (typed in via the *Graphical User Interface*) is sent to the *Dialogue Manager*. The *Dialogue Manager* keeps track of the current stage in the dialogue and controls the use of several submodules. Before handing back control (together with the relevant data) to the *Toplevel*, the input is first sent to the *Natural Language Frontend* which returns a so-called

⁴ <http://www.scoot.co.uk>

⁵ <http://www.bigyellow.com>

⁶ <http://www.bigbook.com>

⁷ <http://www.switchboard.com>

slot-and-filler query. The *Dialogue Manager* then consults the *Query Construction Component*, passing to it the result of the parsing process (possibly modified depending on the *dialogue history*, etc). The purpose of the *Query Construction Component* is to transform the input into a *database query* (making use of the *Backend* and possibly the *World Model*), to query the *Backend* and to return the retrieved addresses (and some database information) to the *Dialogue Manager*. Finally the *Dialogue Manager* hands back control to the *Toplevel* which for example displays the retrieved addresses. It could also put questions to the user which were passed to it by the *Dialogue Manager*, if the database access was not successful (i.e. did not result in a set of addresses). At this stage the cycle starts again.

Figure 1 shows the relationships of the modules in the YPA system.

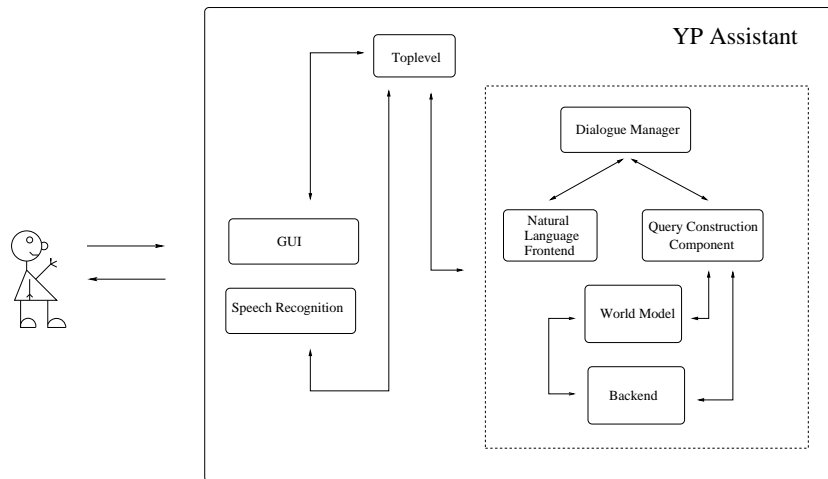


Fig. 1. Architecture of the YPA

We will now describe what each module does by taking the example *I want an Italian restaurant in Colchester that takes Visa* and following, more or less, the sequence of events in one interaction.

3.1 The Graphical User Interface

The *Graphical User Interface* is a form interface that allows a user to enter a query and then calls the *Toplevel*, which returns the content of an HTML page.

The input field is used to state a request for addresses from the *Yellow Pages*. The input query can be any English sentence or phrase or single word. Examples for possible queries are:

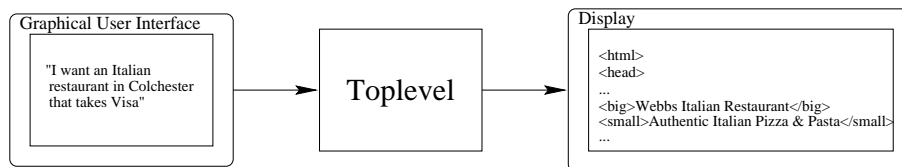
*I want an Italian restaurant in Colchester.
Italian restaurants please!
RESTAURANTS in Colchester*

The *Graphical User Interface* also provides different check boxes for the user to set parameters that influence the strategy of the YPA.

3.2 The Toplevel

The responsibility of the *Toplevel* lies in handling an incoming user request, forwarding the task to the *Dialogue Manager* and passing the returned output back to the *Graphical User Interface* in an appropriate format.

After connecting to the *Dialogue Manager* and waiting for its results, the *Toplevel* returns an HTML-page. This is a sample call:



3.3 The Natural Language Frontend

The *Natural Language Frontend* [15] is called directly by the *Dialogue Manager*. It consists of two components whose overall task is to transfer input strings from the user into slot and filler constructions. These two components, the *Parser* and the *Slot-Filler*, are described in more detail below.

The Parser Parsing methods for handling real NLP applications are being developed all the time. Notably, there are methods such as the LR parsing algorithm [16], phrasal parsing as seen in the SPARKLE project [17] and chunking [18]. It is a primary aim of these systems that they return some linguistically correct parse tree for the input, and have the ability to choose correctly between syntactically (and semantically) ambiguous analyses. For example, the different attachments of prepositional phrases.

For our purposes, it made little difference in the construction of a query if input such as *Italian Restaurant* was seen as an adjective/noun construction or a compound of two nouns. Both terms would be used in the search regardless.

What was required by the *Yellow Pages* domain was not the linguistically correct syntactic tree structure, but the identification of those parts of the input which would allow us access to the meaning of the query, and that this process be a fast one. To that end, we began developing our own parsing process.

The requirement to handle ill-formed input meant that there was no need for a detailed grammar. Instead a simple DCG-like grammar was adopted, since

it was easy both to understand and modify, and a basic bottom-up chart parser was implemented [19].

We wanted to rapidly assign a “skeleton framework” to any input based on the closed class words. Any other words in the input would be given a limited freedom to try out different classes until one allowing a parse could be generated. As soon as one parse tree exists, it is accepted. We make no claim about the resolution of ambiguity on the basis of syntactic structure, but rather let the domain database react to such ambiguities as it sees fit.

All closed class words are listed as *hardwords*, other words were listed as *softwords*, in that we would prefer them to adopt particular word-class behaviour in the majority of situations. For example, the word ‘phone’ (listed in our lexicon as a *softword* verb) as in *I want to phone for a pizza*, compared to *I want to buy a phone*. In the first example we would want it to be a verb, and a noun in the second.

It was stipulated that any word that was not a hardword could also be a noun. This meant that any unknown words were identified as nouns, giving our system the powerful capability of dealing with previously unseen words. The loose nature of our grammar (in that it accepts structures such as prepositional phrases and noun compounds as complete sentences) helps to address a similar problem with unseen grammar constructions.

All this meant that the lexicon could consist of the prepositions and the determiners (as our *hardwords*), and those verbs which we extracted directly from the *Yellow Pages* as our suggested *softwords*, and nothing else.

As an example, here is the result of the parse process in brief for the query *I want an Italian restaurant in Colchester that takes Visa*:

```
[s, [np, [pron, i]] , [vp, [v, want] , [np, [det, an] ,
  [np, [n, italian] , [np, [n, restaurant] ,
    [pp, [prep, in] , [np, [n, colchester] ,
      [pp, [prep, that] , [np, [n, takes] , [np, [n, visa]]]]]]]]]] .
```

Once this brutal parse analysis has been attached to the input, it is passed to the next stage, the slot-filling process.

The Slot-Filling Process The notion of slots and fillers, or frames as they are sometimes called, has been around for some time [20]. They are a way of specifying some semantic interpretation of user input as attributes of slots expressly identified by the domain.

A number of Natural Language Dialogue Systems have used a similar representation, notably Voyager [10], ATIS [21] and RailTel [22]. An important difference between the use of the slot filler mechanism in these systems and in the YPA is the YPA’s preservation of some syntactic structure.

The input parse tree is broken, where possible, into *subject, verb, object, modifiers*. Using this method, the parse tree in the example above would be split into the flatter structure:

```

subject   i
verb      want
object    an italian restaurant
modifiers in colchester, that takes visa

```

Once the input is broken in this way, it is passed to the *domain dependent pragmatic analysis*. At this point, it becomes necessary to know about the behaviour and requirements of the domain.

Some mappings are straightforward. For example in this domain, as it is usually not necessary to know the subject information, this is discarded. The verb information maps directly into the transaction slot, and generally speaking the object maps into the goods slot.

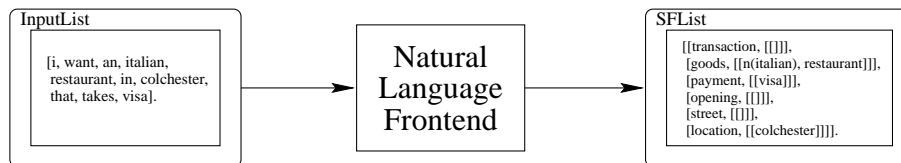
With malformed input such as lone noun phrases (for example *parachuting centre*), it is assumed that the user has entered only the object and any modifiers.

Furthermore, domain-dependent knowledge is used to remove stop words from our slots - words such as *address* or *phone number* which we had identified through the construction of the back-end database as bearing little information content with relation to the domain. This gives us a final output structure for our example of:

```

[[transaction, [[]]],
 [goods, [[n(italian), restaurant]]],
 [payment, [[visa]]],
 [opening, [[]]],
 [street, [[]]],
 [location, [[colchester]]]]

```



3.4 The Dialogue Manager

The interaction between user and machine can be roughly summarized as a filling of different slots in a *slot and filler* query with the information extracted from the user input, so that the database access finally retrieves an acceptable set of entries. In this process, the *Dialogue Manager* is the vital control module. Each stage in the dialogue with the user is characterized as a *dialogue state*. Every time the *Dialogue Manager* is called it performs the following tasks:

- calling the *Natural Language Frontend*
- evaluating the parsed input and determining the dialogue state (updating the *dialogue history*)
- performing all actions corresponding to the state transition.

The overall architecture of the *Dialogue Manager* is very similar to the *PURE* system [23]. The user input is passed to the *Dialogue Manager*, which calls the *Natural Language Frontend* and, depending on the result, decides whether for instance the database should be accessed (calling the *Query Construction Component* and passing the current *slot and filler* query as an argument) or whether the dialogue should be restarted.

Our *Dialogue Manager* consists of a *Core Dialogue Manager* which is the domain-independent heart of the system and an extension (also domain-independent) which adds the basic functionality of a dialogue manager to a frame-based approach and which is called the *Default Dialogue Manager*. The administrator has to (1) set interfaces to the *Core Dialogue Manager* and the *Default Dialogue Manager* and (2) customize the system for the specific application.

The general idea about a domain independent basic dialogue manager is to have a core dialogue manager that covers all tasks to be handled by any similar dialogue system without having to access the *database system*. It should detect

- that a user wants to *quit* (or *restart*) the dialogue;
- *meta queries* (where the user asks for some help, etc);
- that a user uttered some *correction*.

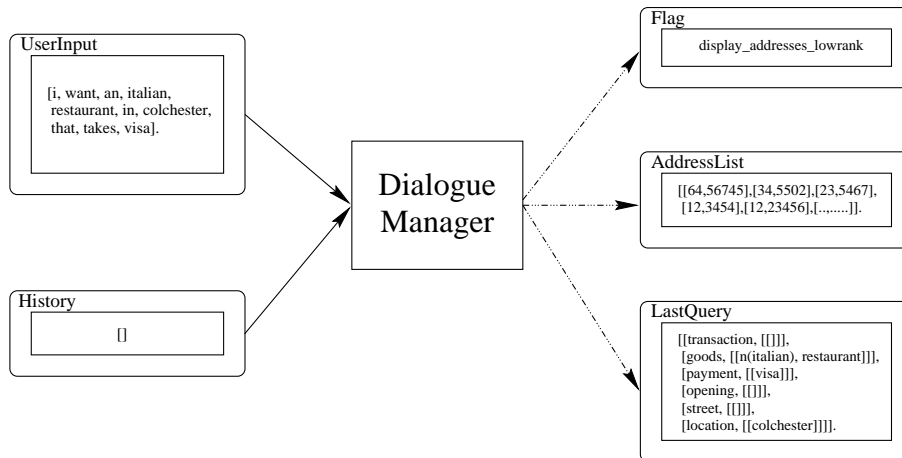
The *Default Dialogue Manager* is this core engine expanded by adding coverage of the other states that can occur in a general spoken dialogue system:

- *mandatory slots* (which must be filled in order to submit a query to the database system) are not filled;
- *unknown concepts* occurred in the input;
- some *inconsistency* occurred;
- a database access was *successful*;
- a database access results in *too many matches*;
- a database access results in *too few matches*.

This outline compares to the two-layered dialogue architecture in Agarwal [23], where the *Default Dialogue Manager* covers the upper layer of dialogue states, and where customization may refine those and add a second (domain-dependent) layer. Differences, however, are the set of dialogue states and the distinction made between the various possible states.

In order to keep the *Dialogue Manager* truly generic we do not assume anything about the structure of the query except that it is some sort of *slot and filler* list. In the YPA, we customized its current *Dialogue Manager* by defining the appropriate interfaces in the setup files.

Here is a sketch of the data flow in the *Dialogue Manager* in the case of our initial example:

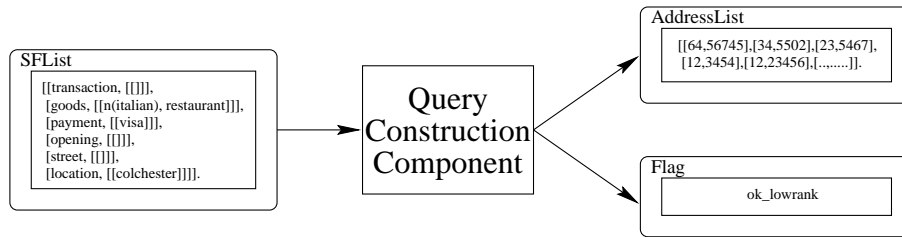


3.5 The Query Construction Component

The structure passed to the *Query Construction Component* is a *slot-and-filler* query. We believe that a large number of sample dialogues involve the construction of *slot-and-filler* structures, and there is nothing special about the slots (*goods*, *transaction*, *location*) used here apart from the fact that the *Query Construction Component* in this system knows how to convert them into queries suitable to be used with the indices in the *Backend*. The task is to match this query to a set of addresses by consulting different sources of knowledge, namely the transformed *Yellow Pages* data (part of the *Backend*) and knowledge sources which can be summarized as the *World Model*. While the *Backend* supplies indices as well as ranking values, the shallow *World Model* delivers information which can be employed on the *Backend* (e.g. for query expansion). It is therefore the task of the *Query Construction Component* to evaluate the various information sources (e.g. indices *versus* ranking values) and retrieve a set of addresses from the *Backend* if possible.

The constructed query is sent to the address database. If this results in a set of addresses (up to a maximum number defined by the administrator), then the *Query Construction* is finished. Otherwise there is a general strategy of successive relaxation of the query. A query that resulted in no matching addresses would for example be relaxed by ignoring slots which are not as important as others (e.g. *opening hours* as opposed to *goods & services*) or by exploiting syntactic information (prepositional phrases could be ignored). This all depends on how the component is set up. If too many addresses are retrieved, then the query will be further constrained if possible. If after query modification there is still no set of addresses, then an appropriate flag is passed back to the *Dialogue Manager*.

For the *Italian restaurant* example the input and output data could look like this:



3.6 The Backend

Figure 2 reflects the data flow in the process of constructing the databases that form the *Backend* of the YPA.

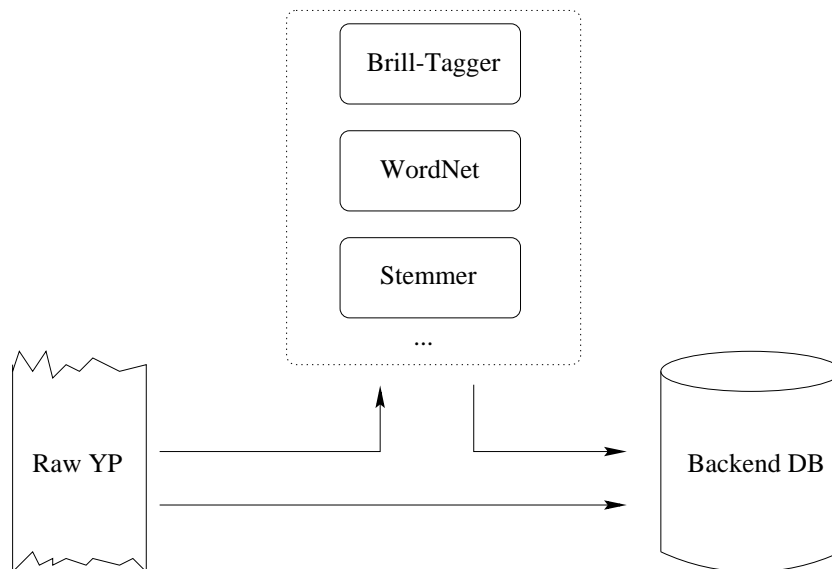


Fig. 2. Extraction of the YPA-Backend

The data extraction and transformation takes place in several steps.

The same extraction techniques result in significant differences in the results when applied to different parts of the input.

There are some general techniques we apply to all the data.

Firstly, for part-of-speech tagging we use the Brill tagger [24,25] without training (using the supplied *lexical rule* and *contextual rule* files of the *Wall Street Journal Corpus* and the lexicon of both *Wall Street Journal* and *Brown Corpus*). This tagger is especially good for our purpose as we do need a contex-

tual tagger, and one that is robust enough. Furthermore the tagging follows the *Penn Treebank* guidelines [26] which makes the results comparable.

Secondly, we use *WordNet* [2] for indexing the keywords. To do this, we use the *WordNet* interface that performs morphological reduction to base forms. We then apply a stemmer to further reduce the base forms delivered by *WordNet*. The result of the stemming does not have to be a proper lexicon entry [27]. However, we can still make use of synonyms as provided by *WordNet*.

Thirdly, the *ID* for each entry in the data is the unique line number where this entry starts. Hence we automatically have a key for most of the relational tables to be created.

The *Backend* contains the information from the raw *Yellow Pages*. There are three subcomponents that form the *Backend*:

- the *Relational Database*, which contains all the information extracted from the raw data file (the actual addresses, indexes, etc);
- the *Information Retrieval Database*, which contains information about the extracted data (occurrences, term frequencies, etc);
- the *Language Module*, which provides base forms for any word form.

The Relational Database The main purpose of the *Relational Database* is to represent the addresses that were extracted from the *Yellow Pages*. This is done fully automatically by applying a set of *UNIX* scripts to the *YP data file* exploiting the record structure of the entries as described in De Roeck et al [1].

In the data file there are entries, whose type is determined by the values of certain fields in the record structure. These entries can be address entries as well as heading entries or reference entries of various types. What is used as the address database is the set of *free entries*, *line entries* and *semi display entries*.

Tables exist for address entries (complete addresses, company names, keyword indexes for the free text of addresses, keyword indexes for the company names, etc) and for headings used in the *Yellow Pages* (complete headings, keyword indexes, *see*-references and *see-also*-references).

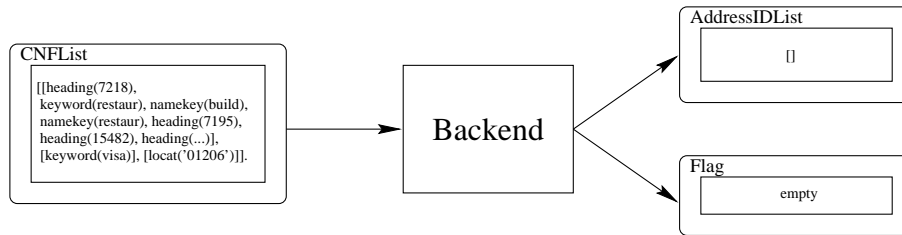
While most of the relations are obvious something must be said about the *see*- and *see-also*-references. Each of the addresses fall under a unique heading. However, some headings are not followed by any address but instead contain a pointer to a set of other headings. This reference is called *see*-reference (for example the heading *Abortion Advice* makes a reference to *Clinics* and *Pregnancy Test Services*). The *see-also*-references are similar but there are still addresses listed under the heading (heading *Abattoirs* makes reference to *Horse Slaughterers* but also lists two addresses in the Colchester data file).

Additionally there are tables for relations which are not directly retrieved from the data file but derived by adding the information contained in the back cover of the *Yellow Pages* together with variations in the usage of town names detected in the actual addresses: *town indexes* and *dialling codes*.

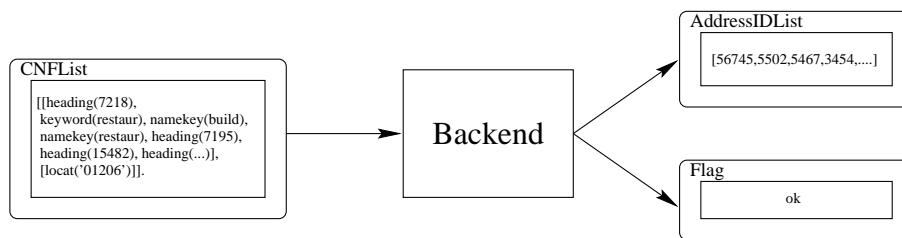
The *town index* relation is used to map variations in the usage of town names onto a unique town index. This town index is used as the first argument in the

dialling code relation, the second argument being the dialling code. For example the town index for the place Clacton-on-Sea is *clacton_on_sea* as it is mostly used in the addresses, but '*clacton on s*' and *clacton-o-s* denote the same location and therefore determining the dialling code for Clacton-on-Sea involves looking up the town index (*clacton_on_sea*) and then consulting the dialling code table with this key.

In the case of the *Italian restaurant* example, we get something like this:



The returned flag *empty* indicates no matching addresses can be found (because there are no Italian restaurants that take Visa). However, it is the task of the *Query Construction Component* to evaluate this output and possibly query the *Backend* again with a modified query. In this case the requirement *keyword(visa)* is relaxed, leading to a revised query:



The Information Retrieval Component The tables in the *Information Retrieval Component* contain *meta information* about the data from the *Yellow Pages*, i.e. information about the distribution of keyword indexes in either the headings, etc. This data is for example used by the *Query Construction Component* to determine the weights for retrieved addresses. These relations express something like: “How many addresses would be retrieved if a certain index would be accessed?” for example:

“If using the keyword index *public_house*, then how many addresses would be retrieved?”

This relation is called *ranking*. The above example would access the table that contains the overall ranking, but the *Query Construction Component* might evaluate the usefulness of more detailed rankings, which are:

- ranking of keyword indexes in the free text of the addresses;

- ranking of keyword indexes in the company names;
- ranking of keyword indexes in the headings;
- overall ranking.

To take the example again, the keyword index *public.house* turns up only in two headings (ranking value 681, since that many addresses would be retrieved) and never in the free text or the name field of the data file (therefore no ranking entry in these two tables). The *overall* ranking appears to be the same as for the heading, but this is normally not the case.

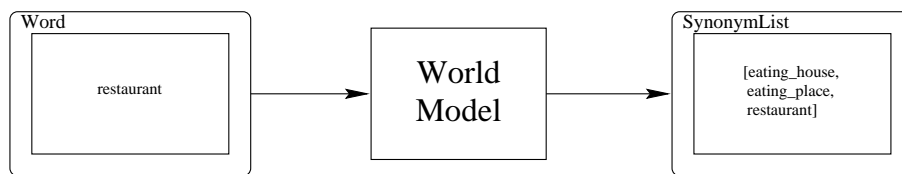
Language Module The *Language Module* provides interfaces for the morphological reduction of *word forms* to *base forms* and for the reduction of *base forms* to *word stems*. These functions are useful when trying to access the *Relational Database* or the *Information Retrieval Component* since the indexes are constructed on the base of the *Language Module*.

The implemented *Language Module* accesses the *WordNet* library. This is of no interest for the interface predicates as these functions can be supplied by any other system, but in this case the *Backend Databases* would have to be rebuilt in order to match the stemmed base forms stored in the index files with the base forms used in the online dialogue.

3.7 The World Model

The function of the *World Model* is to allow query expansion and query modification. This is done either to retrieve addresses or to allow the user to choose various ways of modifying the original input if the query cannot be fired successfully. It contains both domain-independent data (a large lexicon containing various simple hierarchies (*WordNet* [2])) and domain-dependent data (the heading structure from the *Yellow Pages* as well as knowledge acquired by the user, updated via the *YPA AdminTool*). Other sources can be incorporated.

Parameters define the way *world knowledge* is applied. By default, synonyms and hypernyms of query terms are only used when there cannot be found any matching addresses otherwise. But the user can choose to always apply this *world knowledge* or never, or stick to the default.



4 Recent Experiences

The YPA is not a single program but consists of two servers and several scripts which are responsible for the data and control flow between user and main server. It is accessible across the net, and the user interface is through a web browser. Figure 3 shows a screenshot of a reply to a query.

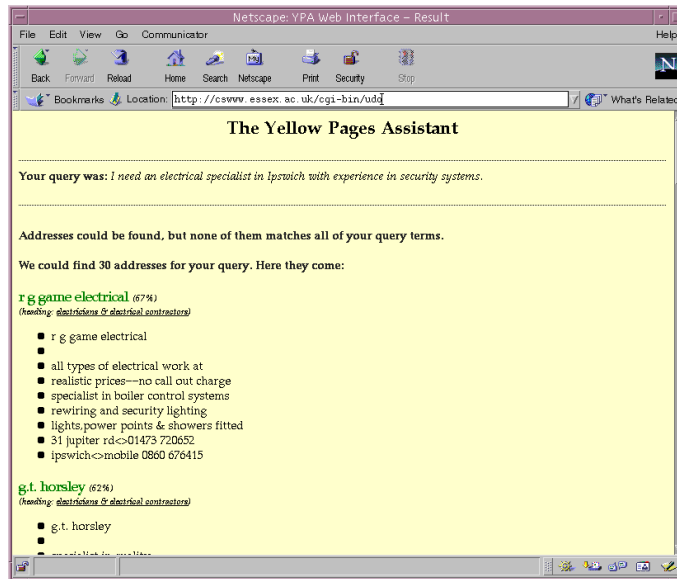


Fig. 3. Screenshot of an online dialogue

4.1 Evaluation

For the evaluation we used the YPA version 0.7 for the Colchester area with about 26,000 addresses.

To date, there have been two evaluations of the YPA system, the first focusing on technology issues, the second on the precision of the recalled addresses.

The first evaluation, that of the technology, reflects mostly on the behaviour of our robust *Natural Language Frontend* process.

The evaluation of such parsing strategies is an art in itself. There have been a large number of suggested methods for evaluating parsers for the purpose of guiding and monitoring the development of a parsing system (see Carroll et al [28] for a good survey).

For the technology evaluation we used the method of *coverage*, where we calculate the percentage of sentences assigned an analysis from an unannotated corpus.

During a two-week period, members of BT's Intelligent Systems Research Group had access to the YPA via the World Wide Web. They were given an outline of the information the system contained, but not told how the system should be queried. Over this time, some 238 *unique* queries were collected by the administrators of the system. These queries were used to conduct the technology-focused evaluation of the YPA system. Each query was evaluated to see if it presented a list of addresses to the user, either first time or as the result of some dialogue with the system. If a list of addresses was not returned, we recorded

the reason for this, and presented the figures as percentages of the total number of queries.

90% were successful from the parser point of view, although of those 22% failed at some later point in the system (incorrect query construction or spelling errors - note that these did not cause a parse failure, but failed to match information in the database). 10% were recorded as a *Frontend* failure - where either queries failed the parse completely, or information was misplaced, and put in incorrect slots.

This method of evaluation is very crude, given that the rules in the grammar allow a variety of parse trees, which might not necessarily be correct. Although linguistic correctness was not a requirement, it is equally important that we have some form of correct structural identification, as we aim to preserve some structural information throughout the slot-filling procedure.

It is often difficult to assess the performance of individual components of systems such as these (see Dybkjaer et al [29]), because the functionality of individual modules is often dependent on other modules within the system.

We also report on the second evaluation, that of the system as a whole, as this gives us an indication of the success of our goal to improve precision.

For the precision evaluation, we used a corpus of 75 queries, collected from the Talking Pages call centre in Bristol. Of these 75 queries, 62 (83%) asked for information about addresses contained within our sample set (that is, existed within the Colchester area). For those, we had a value of 74% of addresses returned being relevant to the input query.

The issue of how to deal with 'negative return queries', that is how to satisfactorily inform the user that no answers can be found, is one that we wish to address in the next stage of dialogue management development, and is probably a major factor in user satisfaction with any system.

These are the results of a very simplistic evaluation. One aspect for example is not reflected at all, that is the order and ranking values of the advertisements in a successful user query.

We continue with our evaluations of the YPA. At the moment we are most concerned about a more detailed evaluation, taking into account *recall* as well as *precision* values.

5 Conclusions and Future Work

We continue work on the YPA system. One of the interesting problems is to extract more structural information from richer data files. At the same time this will influence the other components in particular the *Query Construction Component* and the *Dialogue Manager*, because a more varied *Backend* database allows a refined query construction and a more user-friendly dialogue.

We also noticed that the function of the *Dialogue Manager* changed considerably in the process of the overall development of the YPA. More and more tasks which were initially located in the *Dialogue Manager* have been implemented as

customizations of the *Query Construction Component*. It could well be that in future we will merge those two components to just one.

We are also making the developed data extraction tools more generic. We have recently created a similar system which can extract information from an advertisement such as the one below but still store it in the same *Backend* structure.

PRODUCTS AND SERVICES

- * Authentic American themed restaurant and bar.
- * International menu offering over 60 American, Tex Mex, Cajun and Italian dishes
- * Specialities include BBQ ribs, Fajitas and our famous burgers
- * Famous bar offering over 500 different cocktails and beers from around the world
- ...

PAYMENT METHODS

Cash, Cheque, Visa, Mastercard, American Express, Delta.

OPENING HOURS

12 Midday to 11.30pm - 7 days a week
Open Bank Holidays and New Years Day

...

It proved to be very simple to convert the new data files once the set of scripts had been developed. Minimal customization of the scripts was needed.

A major task for the future is a deep evaluation of the system which involves another user trial once we have developed a framework for this.

6 Acknowledgements

This work has been funded by a contract from ISR group at BT's Adastral Park, Martlesham Heath, UK. Most of the data was provided by Yellow Pages. The authors want to thank K C Tsui and Wayne Wobcke for their helpful comments.

References

1. De Roeck A., Kruschwitz U., Neal P., Scott P., Steel S., Turner R. and Webb N.: 'YPA - an intelligent directory enquiry assistant'. BT Technology Journal, Vol. 16(3), pp. 145-155 (1998).
2. Miller G.: 'Wordnet: An on-line lexical database'. International Journal of Lexicography, Vol. 3(4) (1990). (Special Issue).
3. Nederhof M.-J., Bouma G., Koeling R. and van Noord G.: 'Grammatical analysis in the OVIS spoken-dialogue system'. In 'Proceedings of the ACL/EACL Workshop on "Interactive Spoken Dialog Systems: Bringing Speech and NLP Together in Real Applications"' (Madrid) (1997).

4. Aust H., Oerder M., Seide F. and Steinbiss V.: 'The Philips automatic train timetable information system'. *Speech Communication*, Vol. 17, pp. 249–262 (1995).
5. Peckham J.: 'A new generation of spoken dialogue systems: results and lessons from the SUNDIAL project'. In 'Proceedings of the 3rd European Conference on Speech Communication and Technology' (Berlin, Germany), pp. 33 – 40 (1993).
6. McGlashan S., Fraser N., Gilbert N., Bilange E., Heisterkamp P. and Youd N.: 'Dialogue Management for Telephone Information Systems'. In 'Proceedings of the International Conference on Applied Language Processing' (Trento, Italy) (1992).
7. Allen J., Schubert L., Ferguson G., Heeman P., Hwang C., Kato T., Light M., Martin N., Miller B., Posesio M. and Traum D.: 'The TRAINS project: a case study in building a conversational planning agent'. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 7, pp. 7–48 (1995).
8. Wahlster W.: 'Verbmobil: Translation of Face-to-Face Dialogues'. In 'Proceedings of the 3rd European Conference on Speech Communication and Technology' (Berlin, Germany), pp. 29–38 (1993).
9. Zue V., Glass J., Goodine D., Leung H., Phillips M., Polifroni J. and Seneff S.: 'The VOYAGER Speech Understanding System: Preliminary Development and Evaluation'. In 'Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing' (Cambridge, MA, USA), pp. 73–76 (1990).
10. Zue V.: 'Toward Systems that Understand Spoken Language'. *IEEE Expert Magazine*, Vol. February, pp. 51–59 (1994).
11. Glass J., Flammia G., Goodine D., Phillips M., Polifroni J., Sakai S., Seneff S. and Zue V.: 'Multilingual Spoken-Language Understanding in the MIT VOYAGER System'. *Speech Communication*, Vol. 17, pp. 1–18 (1995).
12. Flank S.: 'A layered approach to NLP-based Information Retrieval'. In 'Proceedings of the 36th ACL and the 17th COLING Conferences' (Montreal), pp. 397–403 (1998).
13. Woods W. A.: 'Conceptual Indexing: A Better Way to Organize Knowledge'. Technical Report SMLI TR-97-61, Sun Microsystems Laboratories, Mountain View, CA (1997).
14. Ambroziak J. and Woods W. A.: 'Natural Language Technology in Precision Content Retrieval'. In 'Proceedings of the 2nd Conference on Natural Language Processing and Industrial Applications (NLP-IA)' (Moncton, Canada), pp. 117–124 (1998).
15. Webb N., De Roeck A., Kruschwitz U., Scott P., Steel S. and Turner R.: 'Natural Language Engineering: Slot-Filling in the YPA'. In 'Proceedings of the Workshop on Natural Language Interfaces, Dialogue and Partner Modelling (at the Fachtagung für Künstliche Intelligenz KI'99)' (Bonn, Germany) (1999). http://www.ikp.uni-bonn.de/NDS99/Finals/3_1.ps.
16. Tomita M.: 'Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems'. Kluwer Academic (1985).
17. Briscoe T., Carroll J., Carroll G., Federici S., Montemagni G. G. S., Pirrelli V., Prodanof I., Rooth M. and Vannocchi M.: 'Phrasal Parser Software - Deliverable 3.1'. <http://www.ilc.pi.cnr.it/sparkle.html>, (1997).
18. Abney S.: 'Parsing by chunks'. In 'Principle-Based Parsing', S. A. R. Berwick and C. Tenny, Eds. Kluwer Academic Publishers (1991).
19. Gazdar G. and Mellish C.: 'Natural Language Processing in PROLOG: An Introduction to Computational Linguistics'. Addison Wesley (1989).
20. Minsky M.: 'A Framework for Representing Knowledge'. In 'The Psychology of Computer Vision', P. H. Winston, Ed. McGraw-Hill, New York, pp. 211–277 (1975).

21. Bannacef S. K., Bonneau-Maynard H., Gauvain J. L., Lamel L. and Minker W.: 'A Spoken Language System for Information Retrieval'. In 'Proceedings of the International Conference on Speech and Language Processing' (Yokohama, Japan) (1994).
22. Bannacef S. K., Devillers L., Rosset S. and Lamel L.: 'Dialog in the RAILTEL Telephone-Based System'. In 'Proceedings of the International Conference on Speech and Language Processing' (Philadelphia), pp. 550–553 (1996).
23. Agarwal R.: 'Towards a PURE Spoken Dialogue System for Information Access'. In 'Proceedings of the ACL/EACL Workshop on "Interactive Spoken Dialog Systems: Bringing Speech and NLP Together in Real Applications"' (Madrid), pp. 90–97 (1997).
24. Brill E.: 'A simple rule-based part of speech tagger'. In 'Proceedings of the Third Conference on Applied Natural Language Processing, ACL' (Trento, Italy) (1992).
25. Brill E.: 'Some advances in rule-based part of speech tagging'. In 'Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)' (Seattle, Wa.) (1994).
26. Santorini B.: 'Part-of-speech tagging guidelines for the Penn Treebank Project'. Technical report MS-CIS-90-47, Department of Computer and Information Science, University of Pennsylvania (1990).
27. Strzalkowski T.: 'Natural Language Information Retrieval: TREC-4 Report'. In 'Proceedings of the Fourth Text Retrieval Conference (TREC-4)' (NIST Special Publication 500-236) (1996).
28. Carroll J., Briscoe T. and Sanfilippo A.: 'Parser evaluation: a survey and a new proposal'. In 'Proceedings of the 1st International Conference on Language Resources and Evaluation' (Granada, Spain), pp. 447 – 454 (1998).
29. Dybkjær L., Bernsen N. O., Carlson R., Chase L., Dahlbäck N., Failenschmid K., Heid U., Heisterkamp P., Jönsson A., Kamp H., Karlsson I., v. Kuppevelt J., Lamel L., Paroubek P. and Williams D.: 'The DISC approach to spoken language systems development and evaluation'. In 'Proceedings of the 1st International Conference on Language Resources and Evaluation' (Granada, Spain), pp. 185 – 189 (1998).